Sprawozdanie WMM Laboratorium 5

Miłosz Andryszczuk 331355

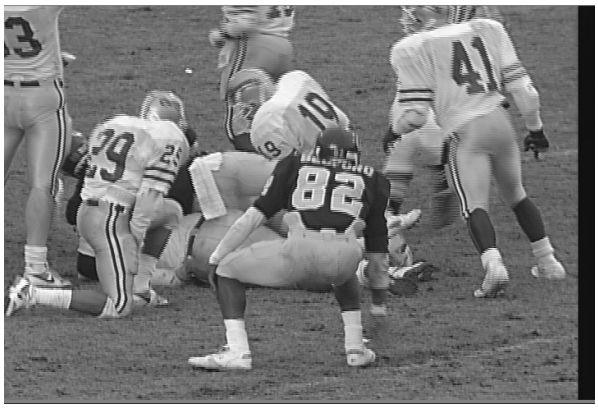
Grupa 104 Piątek 14:15

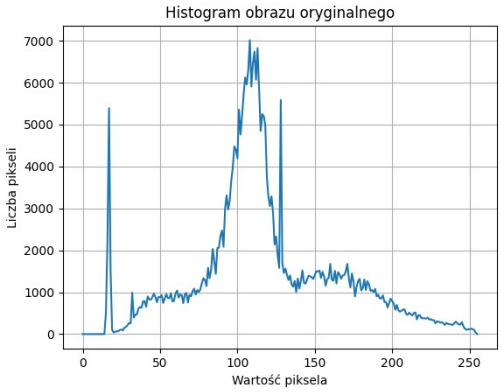
numer_obrazu = numer_indeksu % liczba_obrazow = 331355 % 36 = **11**

Obraz monochromatyczny

Zadanie 1

W celu policzenia entropii obrazu monochramatycznego w pierwszej kolejności obliczono histogram obrazu.

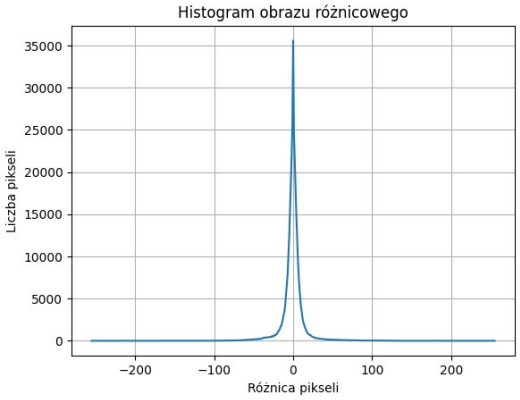




Entropia oryginału wynosi w przybliżeniu 7.29.
Stosunkowo wysoka entropia świadczy o dużej różnorodności pikseli i obecności wielu unikalnych informacji. Histogram pokazuje rozkład jasności skoncentrowany w kilku obszarach, co oznacza, że mimo tej różnorodności występują też często powtarzające się wartości.

Zadanie 2

Obraz różnicowy – entropia = 5.20

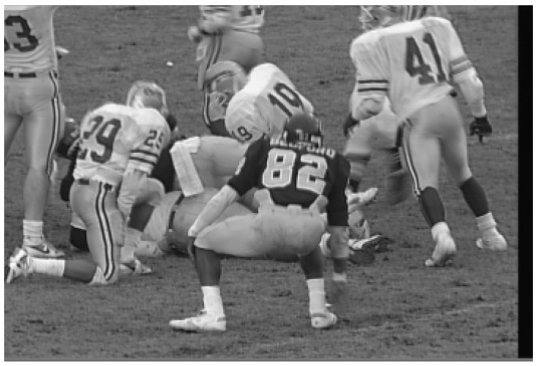


Obraz różnicowy zawiera wyraźnie mniej zróżnicowanych informacji niż obraz oryginalny, o czym świadczy niższa entropia wynosząca 5.20 bpp w porównaniu do 7.29 bpp dla oryginału. Na obrazie różnicowym widoczne są głównie kontury, a tło oraz jednorodne fragmenty zostały mocno zredukowane, co skutkuje dużą liczbą pikseli o wartości bliskiej zeru. Potwierdza to histogram, który w przypadku obrazu różnicowego ma bardzo wyraźny, wąski pik w okolicach zera, natomiast histogram oryginalnego obrazu był znacznie bardziej rozłożony, pokazując większe zróżnicowanie jasności w całym zakresie.

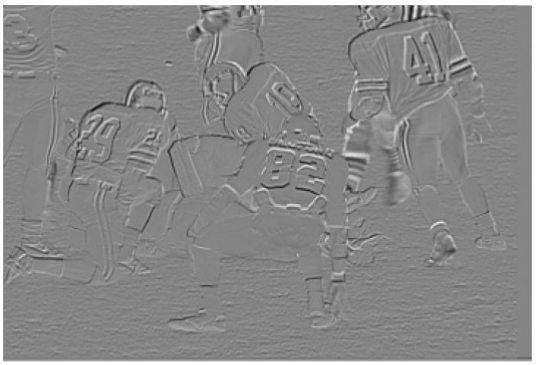
Zadanie 3
Transformata DWT

Poszczególne pasma DWT:

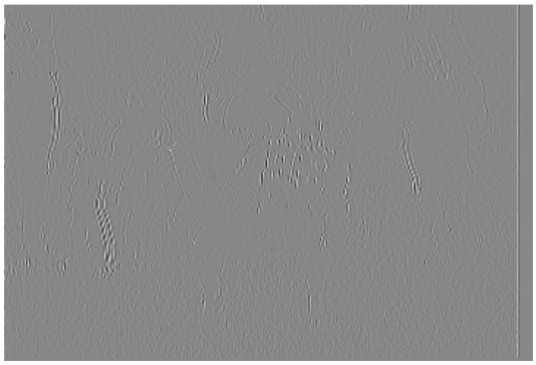
LL



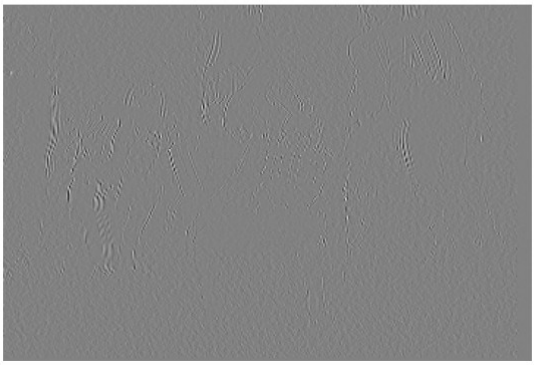
LH



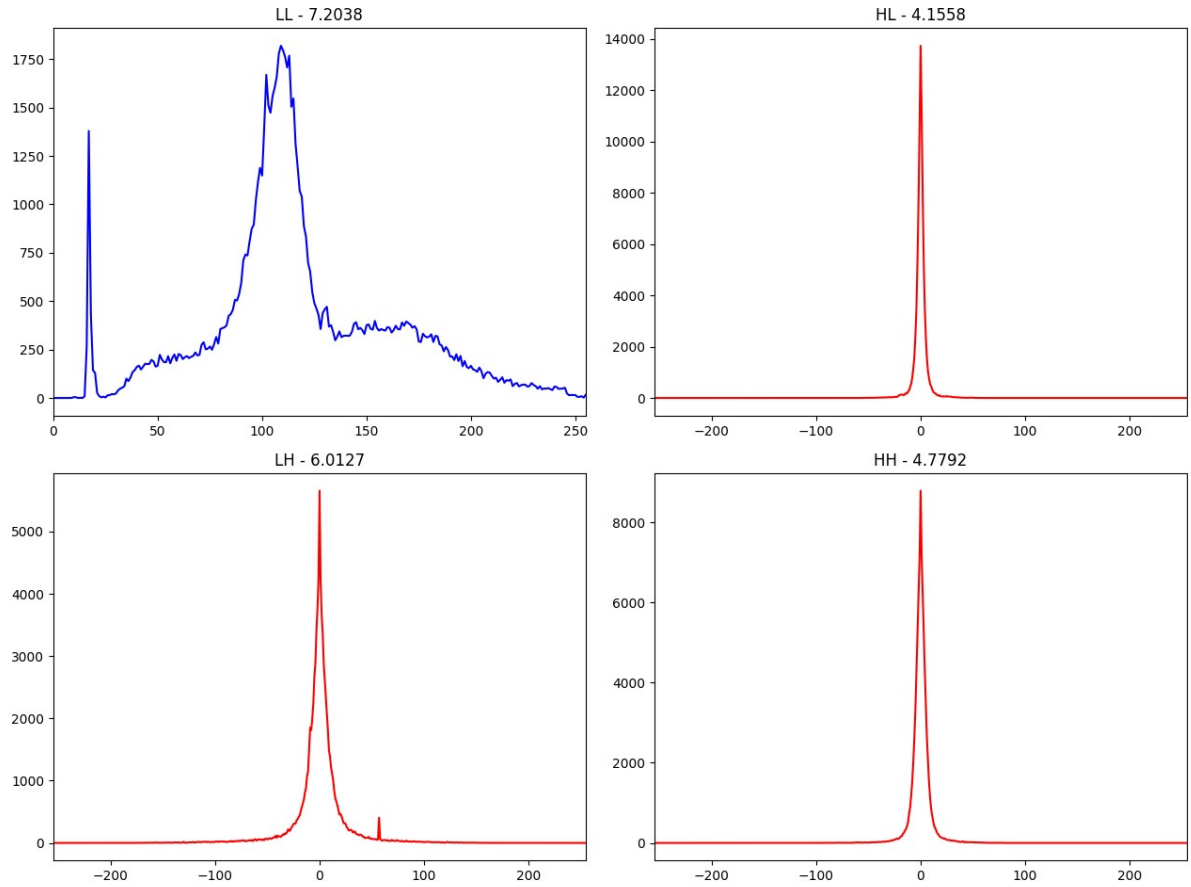
HL



HH



Histogramy pasm DWT razem z ich entropią:



Na podstawie analizy pasm DWT można zauważyć, że największą ilość informacji zawiera pasmo LL, które zachowuje ogólną strukturę obrazu – wygląda jak rozmyta wersja oryginału. Pozostałe pasma (LH, HL, HH) zawierają głównie szczegóły: LH – poziome, HL – pionowe, HH – skośne.

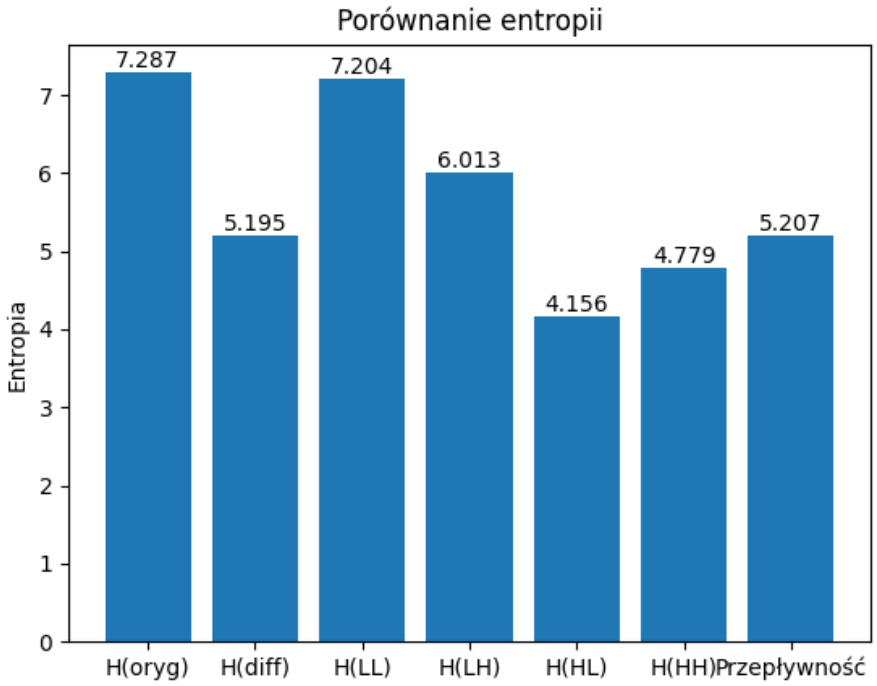
Porównując histogramy pasm DWT widać, że histogram LL jest szeroki, co świadczy o dużej różnorodności pikseli, natomiast histogramy LH, HL i HH są ostro skupione wokół zera, co sugeruje małą zmienność danych w tych pasmach. Potwierdza to także entropia – LL ma wartość 7.20 bitów/piksel, co jest zbliżone do entropii obrazu oryginalnego (7.29), natomiast pozostałe pasma mają znacznie niższe wartości: LH – 6.01, HL – 4.15, HH – 4.77.

W porównaniu do obrazu różnicowego, który miał entropię 5.20 i histogram skoncentrowany wokół zera, pasma LH, HL i HH wyglądają podobnie – również skupione wartości i niska entropia. Oznacza to, że zarówno kodowanie różnicowe, jak i transformacja falkowa pozwalają na bardziej uproszczoną reprezentację obrazu, ponieważ wykorzystują podobieństwa między sąsiadującymi pikselami.

Zadanie 4

Oryginalny obraz zapisano za pomocą kodera PNG, a następnie obliczono jego przepływność (liczba bitów przypadająca na jeden piksel). Przepływność wynosi 5.2075.

Porównanie entropii poszczególnych obrazów i przepływności:



Przepływność wynikająca z kompresji PNG (5.207 bpp) jest bardzo zbliżona do entropii obrazu różnicowego (5.195 bpp), co sugeruje, że PNG – podobnie jak kodowanie różnicowe – korzysta z zależności pomiędzy sąsiednimi pikselami. Przepływność jest także niższa od entropii pasma LL (7.204 bpp), ale wyższa niż entropie pozostałych pasm DWT, które zawierają mniej informacji. Pokazuje to, że PNG kompresuje obraz efektywnie, choć nie osiąga tak niskiego poziomu uproszczenia jak pasma DWT.

Warto jednak zwrócić uwagę na to, że przepływność po kompresji PNG jest niższa niż entropia obrazu oryginalnego. Jest to możliwe, ponieważ entropia liczona na podstawie histogramu nie uwzględnia zależności kontekstowych, które wykorzystuje PNG – takich jak powtarzające się wzorce w sąsiednich pikselach. Kompresor PNG stosuje predykcję i algorytm DEFLATE, dzięki czemu przepływność może być niższa niż obliczona entropia obrazu.

Obraz barwny

Zadanie 1

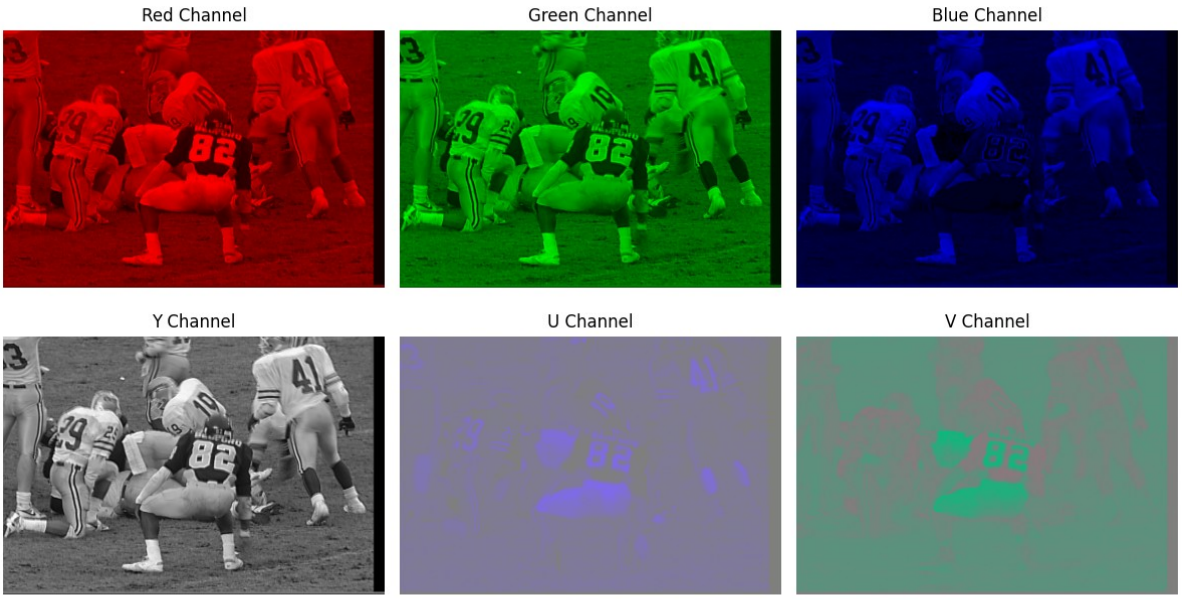
Wyznaczono histogramy i obliczono entropię dla składowych RGB obrazu barwnego.

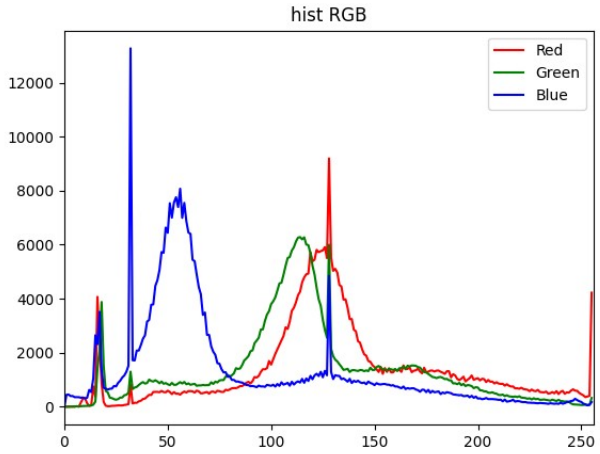
H(R) = 7.3340 H(G) = 7.3334 H(B) = 7.1123 H_śr = 7.2599

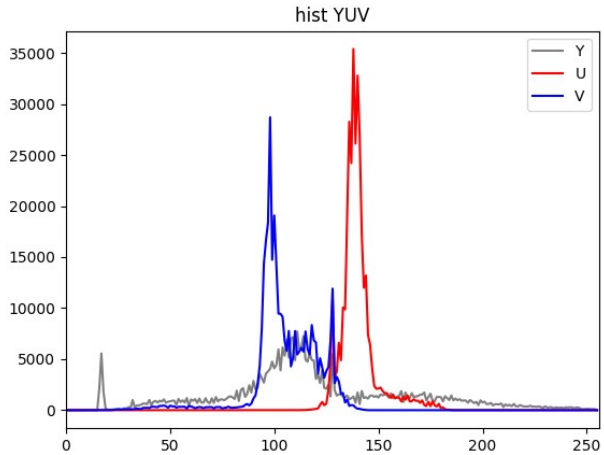
Zadanie 2

Entropia dla poszczególnych składowych YUV i średnia:

H(Y) = 7.2658 H(U) = 4.7294 H(V) = 5.5267 H_śr = 5.8406







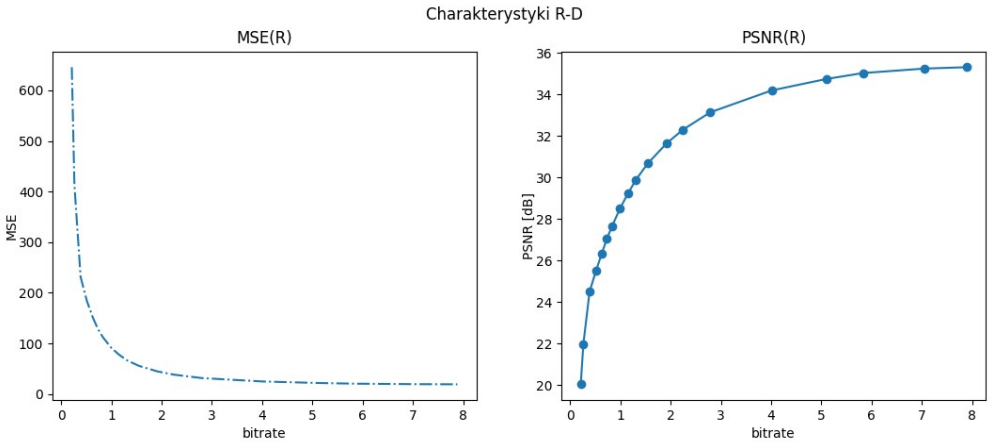
Entropie składowych RGB są bardzo zbliżone i wysokie (około 7.1–7.3), co oznacza dużą różnorodność wartości pikseli w każdym z kanałów. Natomiast w przestrzeni YUV, tylko kanał Y (jasność) ma porównywalnie wysoką entropię, natomiast kanały U i V mają znacznie niższe wartości (odpowiednio ok. 4.7 i 5.5).

Widoczne wyniki entropii bardzo dobrze pokrywają się z tym, co widać zarówno na histogramach, jak i na samych wizualizacjach poszczególnych kanałów. Kanały R, G i B mają podobnie wysokie entropie, co zgadza się z dużą różnorodnością barw i intensywności – potwierdza to histogram, gdzie rozkłady są szerokie i zróżnicowane. W przestrzeni YUV największą entropię ma kanał Y (jasność), który zawiera większość struktury obrazu – co widać na zdjęciu jako wyraźny, szczegółowy obraz w odcieniach szarości. Natomiast kanały U i V (chrominancja) mają znacznie mniejszą entropię – co pokrywa się z histogramem (wąskies rozkłady) i z obrazkami, gdzie występuje niewielka zmienność i dominują rozmyte kolory.

Niższa entropia kanałów U i V wynika z tego, że zawierają one informację o barwie (chrominancji), która zmienia się znacznie wolniej w obrazie niż jasność. Oznacza to, że wartości U i V są mniej zróżnicowane, bardziej skupione wokół jednej wartości, co skutkuje mniejszą entropią i potencjalnie lepszą kompresją.

Zadanie 3

Zależność zniekształcenia D od przepływności R dla kodera JPEG:



Wraz ze wzrostem bitrate wartość MSE (średni błąd kwadratowy) szybko maleje, co oznacza mniejsze zniekształcenie. Jednocześnie PSNR (stosunek sygnału do szumu) rośnie, co oznacza lepszą jakość obrazu. Przy wyższych bitrate’ach zmiany są coraz mniejsze – jakość obrazu poprawia się wolniej, a zyski z większego bitrate’u stają się mniej istotne, ponieważ najważniejsze szczegóły zostały już zachowane i kodowane są już tylko drobne detale.

Subiektywne porównanie stopni kompresji:

Quality 1, Bitrate: 0.21

Quality 25, Bitrate: 0.73

Quality 5, Bitrate: 0.26

Quality 50, Bitrate: 1.15

Quality 15, Bitrate: 0.51

Quality 95, Bitrate: 4.01

Na podstawie wizualnej oceny obrazów zrekonstruowanych przy różnych wartościach jakości (quality) i odpowiadających im przepływnościach (bitrate), można wyróżnić następujące 3 grupy jakości:

Bitrate < 0.3 (np. quality 1 i 5):

Obrazy prezentują bardzo widoczne zniekształcenia. Jakość obrazów w tej grupie jest nie do przyjęcia (mimo to jest możliwość odczytania chociażby numerów na koszulkach).

Bitrate 0.3–0.7 (np. quality 15 i 25):

W tej grupie zniekształcenia nadal są widoczne, ale jest to znaczna poprawa jakości pomimo niewielkiej zmiany bitrate. Jakość określić można jako średnią do dobrej.

Bitrate > 1 (np. quality 50 i 95):

Obrazy w tej grupie prezentują bardzo dobrą lub doskonałą jakość. W przypadku quality 95 trudno gołym okiem odróżnić obraz od oryginału. Quality 50 również daje obraz bardzo dobrej jakości, a używa zaledwie 1.15 bita na piksel.

Zwiększenie bitrate znacząco poprawia jakość obrazu, ale powyżej pewnego poziomu (ok. 1 bpp) dalszy wzrost bitrate nie przynosi już istotnej poprawy jakości wizualnej. Największy skok jakościowy następuje w zakresie od około 0.2 do 1 bpp.

Obliczono bitrate dla barwnego obrazu skompresowanego za pomocą PNG. Wyniósł on 15.65 bpp, co oznacza, że mimo bezstratnej kompresji PNG, format ten i tak skutecznie zmniejsza rozmiar obrazu w porównaniu do nieskompresowanego RGB (24 bpp). Jednak JPEG osiąga znacznie niższy bitrate – nawet w okolicach 1 bpp – oferując przy tym nadal dobrą jakość wizualną. W efekcie JPEG jest znacznie bardziej efektywny pod względem oszczędzania miejsca, ale odbywa się to kosztem utraty części danych obrazu (kompresja stratna).

Kod źródłowy

Na następnych stronach przedstawiono kod do powyższych zadań z podziałem na obraz monochrmatyczny i obraz barwny.

Obraz monochromatyczny

```
import os
import cv2
import numpy as np
from matplotlib import pyplot as plt

def cv_imshow(img, img_title="image"):
    if (img.dtype == np.float32) or (img.dtype == np.float64):
        img_ = img / 255
    elif img.dtype == np.int16:
        img_ = img * 128
    else:
        img_ = img
    cv2.imshow(img_title, img_)
    cv2.waitKey(1)

def calc_entropy(hist):
    pdf = hist / hist.sum()
    entropy = -sum([x * np.log2(x) for x in pdf if x != 0])
    return entropy

def get_histogram_and_entropy(img, size=255):
    hist = cv2.calcHist([img], [0], None, [size], [0, size]).flatten()
    entropy = calc_entropy(hist)
    return hist, entropy

def calc_diff_img(img):
    img_tmp1 = img[:, 1:]
    img_tmp2 = img[:, :-1]

    img_diff = cv2.addWeighted(img_tmp1, 1, img_tmp2, -1, 0, dtype=cv2.CV_16S)
    img_diff_0 = cv2.addWeighted(img[:, 0], 1, 0, 0, -127, dtype=cv2.CV_16S)
    img_diff = np.hstack((img_diff_0, img_diff))

    return img_diff
```

```
def dwt(img):
    maskL = np.array([0.02674875741080976, -0.01686411844287795, -0.07822326652898785,
0.2668641184428723,
        0.6029490182363579, 0.2668641184428723, -0.07822326652898785, -0.01686411844287795,
0.02674875741080976])
    maskH = np.array([0.09127176311424948, -0.05754352622849957, -0.5912717631142470,
1.115087052456994,
        -0.5912717631142470, -0.05754352622849957, 0.09127176311424948])

    bandLL = cv2.sepFilter2D(img,         -1, maskL, maskL)[::2, ::2]
    bandLH = cv2.sepFilter2D(img, cv2.CV_16S, maskL, maskH)[::2, ::2]
    bandHL = cv2.sepFilter2D(img, cv2.CV_16S, maskH, maskL)[::2, ::2]
    bandHH = cv2.sepFilter2D(img, cv2.CV_16S, maskH, maskH)[::2, ::2]

    return bandLL, bandLH, bandHL, bandHH

def calc_bitrate(img):
    cv2.imwrite("compressed_input.png", img)

    height, width = img.shape
    num_pixels = height * width

    file_size_bytes = os.path.getsize("compressed_input.png")

    bitrate = (file_size_bytes * 8) / num_pixels

    return bitrate

def plot_histograms(hist_input, hist_diff):
    plt.figure()
    plt.title("Histogram obrazu oryginalnego")
    plt.plot(hist_input)
    plt.xlabel("Wartość piksela")
    plt.ylabel("Liczba pikseli")
    plt.grid(True)
    plt.savefig("hist_original.png")
    plt.show()
```

```
    plt.figure()
    plt.title("Histogram obrazu różnicowego")
    plt.plot(np.arange(-255, 256, 1), hist_diff)
    plt.xlabel("Różnica pikseli")
    plt.ylabel("Liczba pikseli")
    plt.grid(True)
    plt.savefig("hist_diff.png")
    plt.show()

def plot_histograms_dwt(ll, lh, hl, hh):
    (hist_ll, entropy_ll) = ll
    (hist_lh, entropy_lh) = lh
    (hist_hl, entropy_hl) = hl
    (hist_hh, entropy_hh) = hh

    fig = plt.figure()
    fig.set_figheight(fig.get_figheight()*2)
    fig.set_figwidth(fig.get_figwidth()*2)
    plt.subplot(2, 2, 1)
    plt.plot(hist_ll, color="blue")
    plt.title(f"LL - {entropy_ll:.4f}")
    plt.xlim([0, 255])
    plt.subplot(2, 2, 3)
    plt.plot(np.arange(-255, 256, 1), hist_lh, color="red")
    plt.title(f"LH - {entropy_lh:.4f}")
    plt.xlim([-255, 255])
    plt.subplot(2, 2, 2)
    plt.plot(np.arange(-255, 256, 1), hist_hl, color="red")
    plt.title(f"HL - {entropy_hl:.4f}")
    plt.xlim([-255, 255])
    plt.subplot(2, 2, 4)
    plt.plot(np.arange(-255, 256, 1), hist_hh, color="red")
    plt.title(f"HH - {entropy_hh:.4f}")
    plt.xlim([-255, 255])
    plt.tight_layout()
    plt.savefig("hist_dwt.png")
    plt.show()
```

```
    plt.close('all')

def save_dwt_bands(ll, lh, hl, hh):
    _, axes = plt.subplots(2, 2, figsize=(12, 8))

    axes[0, 0].imshow(ll, cmap='gray')
    axes[0, 0].set_title("LL")
    axes[0, 0].axis('off')

    axes[0, 1].imshow(lh, cmap='gray')
    axes[0, 1].set_title("LH")
    axes[0, 1].axis('off')

    axes[1, 0].imshow(hl, cmap='gray')
    axes[1, 0].set_title("HL")
    axes[1, 0].axis('off')

    axes[1, 1].imshow(hh, cmap='gray')
    axes[1, 1].set_title("HH")
    axes[1, 1].axis('off')

    plt.tight_layout()
    plt.savefig('dwt_pasma.png')

def plot_comparison(data):
    labels = ["H(oryg)", "H(diff)", "H(LL)", "H(LH)", "H(HL)", "H(HH)", "Przepływność"]

    plt.figure()
    plt.title("Porównanie entropii")
    bars = plt.bar(labels, data)
    plt.bar_label(bars, fmt=lambda x: f'{x:.3f}')
    plt.ylabel("Entropia")
    plt.savefig("entropy_comparison.png")
    plt.show()

img = cv2.imread("original.png", cv2.IMREAD_UNCHANGED)
```

```
img_diff = calc_diff_img(img)

ll, lh, hl, hh = dwt(img)

cv_imshow(img, "Obraz oryginalny")
cv_imshow(img_diff, "Obraz roznicowy")
cv_imshow(ll, "LL")
cv_imshow(cv2.multiply(lh, 2), "LH")
cv_imshow(cv2.multiply(hl, 2), "HL")
cv_imshow(cv2.multiply(hh, 2), "HH")
cv2.waitKey(0)
cv2.destroyAllWindows()

hist_input, entropy_input = get_histogram_and_entropy(img)
hist_diff, entropy_diff = get_histogram_and_entropy((img_diff+255).astype(np.uint16), 511)

hist_ll = cv2.calcHist([ll], [0], None, [256], [0, 256]).flatten()
hist_lh = cv2.calcHist([(lh+255).astype(np.uint16)], [0], None, [511], [0, 511]).flatten()
hist_hl = cv2.calcHist([(hl+255).astype(np.uint16)], [0], None, [511], [0, 511]).flatten()
hist_hh = cv2.calcHist([(hh+255).astype(np.uint16)], [0], None, [511], [0, 511]).flatten()

H_ll = calc_entropy(hist_ll)
H_lh = calc_entropy(hist_lh)
H_hl = calc_entropy(hist_hl)
H_hh = calc_entropy(hist_hh)

diff_shifted = (img_diff + 255).astype(np.uint16)
diff_normalized = cv2.normalize(diff_shifted, None, 0, 255, cv2.NORM_MINMAX).astype(np.uint8)
cv2.imwrite("diff.png", diff_normalized)

plot_histograms(hist_input, hist_diff)
plot_histograms_dwt((hist_ll, H_ll), (hist_lh, H_lh), (hist_hl, H_hl), (hist_hh, H_hh))

save_dwt_bands(ll, lh, hl, hh)

bitrate = calc_bitrate(img)
```

```
plot_comparison([entropy_input, entropy_diff, H_ll, H_lh, H_hl, H_hh, bitrate])


print("Entropia obrazu oryginalnego:", entropy_input)
print("Entropia obrazu różnicowego:", entropy_diff)
print(f"H(LL) = {H_ll:.4f} \nH(LH) = {H_lh:.4f} \nH(HL) = {H_hl:.4f} \nH(HH) = {H_hh:.4f} \nH_śr = {(H_ll+H_lh+H_hl+H_hh)/4:.4f}")
print(f"Przepływność PNG (oryginał): {bitrate:.4f} bpp")
```

Obraz barwny

```
import os
import cv2
import numpy as np
from matplotlib import pyplot as plt

def cv_imshow(img, img_title="image"):
    if (img.dtype == np.float32) or (img.dtype == np.float64):
        img_ = img / 255
    elif img.dtype == np.int16:
        img_ = img * 128
    else:
        img_ = img
    cv2.imshow(img_title, img_)
    cv2.waitKey(1)

def calc_entropy(hist):
    pdf = hist / hist.sum()
    entropy = -sum([x * np.log2(x) for x in pdf if x != 0])
    return entropy

def get_histogram_and_entropy(img, size=256):
    hist = cv2.calcHist([img], [0], None, [size], [0, size]).flatten()

    entropy = calc_entropy(hist)
    return hist, entropy
```

```
def show_rgb_channels(r_channel, g_channel, b_channel):
    zeros = np.zeros_like(r_channel)

    r_image = np.stack([r_channel, zeros, zeros], axis=2).astype(np.uint8)
    g_image = np.stack([zeros, g_channel, zeros], axis=2).astype(np.uint8)
    b_image = np.stack([zeros, zeros, b_channel], axis=2).astype(np.uint8)

    fig, axes = plt.subplots(1, 3, figsize=(12, 4))

    axes[0].imshow(r_image)
    axes[0].set_title("Red Channel")
    axes[1].imshow(g_image)
    axes[1].set_title("Green Channel")
    axes[2].imshow(b_image)
    axes[2].set_title("Blue Channel")

    for ax in axes:
        ax.axis("off")

    plt.tight_layout()
    plt.savefig("rgb_channels.png")
    plt.show()

def plot_rgb_histograms(hist_R, hist_G, hist_B):
    plt.plot(hist_R, color="red")
    plt.plot(hist_G, color="green")
    plt.plot(hist_B, color="blue")
    plt.title("hist RGB")
    plt.xlim([0, 256])
    plt.legend(["Red", "Green", "Blue"])
    plt.savefig("rgb_hist.png")
    plt.show()

def show_yuv_channels(img_yuv):
    Y = img_yuv[:, :, 0]
    U = img_yuv[:, :, 1]
    V = img_yuv[:, :, 2]
```

```
    u_vis = np.full_like(img_yuv, 127)
    v_vis = np.full_like(img_yuv, 127)
    u_vis[:, :, 1] = U
    v_vis[:, :, 2] = V

    u_rgb = cv2.cvtColor(u_vis, cv2.COLOR_YUV2RGB)
    v_rgb = cv2.cvtColor(v_vis, cv2.COLOR_YUV2RGB)

    fig, axes = plt.subplots(1, 3, figsize=(12, 4))

    axes[0].imshow(Y, cmap="gray")
    axes[0].set_title("Y Channel")
    axes[1].imshow(u_rgb)
    axes[1].set_title("U Channel")
    axes[2].imshow(v_rgb)
    axes[2].set_title("V Channel")

    for ax in axes:
        ax.axis("off")

    plt.tight_layout()
    plt.savefig("yuv_channels.png")
    plt.show()

def plot_yuv_histograms(hist_Y, hist_U, hist_V):
    plt.plot(hist_Y, color="gray")
    plt.plot(hist_U, color="red")
    plt.plot(hist_V, color="blue")
    plt.title("hist YUV")
    plt.xlim([0, 256])
    plt.legend(["Y", "U", "V"])
    plt.savefig("yuv_hist.png")
    plt.show()

def calc_mse_psnr(img1, img2):
    imax = 255.**2
```

```
    mse = ((img1.astype(np.float64)-img2)**2).sum()/img1.size
    psnr = 10.0*np.log10(imax/mse)
    return (mse, psnr)

def compare_compression_qualities(image, qualities):
    xx = []
    ym = []
    yp = []

    for quality in qualities:
        out_file_name = f"jpeg/out_image_q{quality:03d}.jpg"

        cv2.imwrite(out_file_name, image, (cv2.IMWRITE_JPEG_QUALITY, quality))

        image_compressed = cv2.imread(out_file_name, cv2.IMREAD_UNCHANGED)
        bitrate = 8*os.stat(out_file_name).st_size/(image.shape[0]*image.shape[1])
        mse, psnr = calc_mse_psnr(image, image_compressed)

        xx.append(bitrate)
        ym.append(mse)
        yp.append(psnr)

    fig = plt.figure()
    fig.set_figwidth(fig.get_figwidth()*2)
    plt.suptitle("Charakterystyki R-D")
    plt.subplot(1, 2, 1)
    plt.plot(xx, ym, "-.")
    plt.title("MSE(R)")
    plt.xlabel("bitrate")
    plt.ylabel("MSE", labelpad=0)
    plt.subplot(1, 2, 2)
    plt.plot(xx, yp, "-o")
    plt.title("PSNR(R)")
    plt.xlabel("bitrate")
    plt.ylabel("PSNR [dB]", labelpad=0)
    plt.savefig("quality_comparison.png")
```

```
    plt.show()

def show_jpeg_qualities(image, qualities):
    assert len(qualities) == 6, "The list must contain exactly 6 quality values."

    fig, axes = plt.subplots(3, 2, figsize=(10, 11))

    for i, q in enumerate(qualities):
        filename = f"jpeg/out_image_q{q:03d}.jpg"
        cv2.imwrite(filename, image, [cv2.IMWRITE_JPEG_QUALITY, q])
        img_rgb = cv2.cvtColor(cv2.imread(filename), cv2.COLOR_BGR2RGB)
        bitrate = 8*os.stat(filename).st_size/(image.shape[0]*image.shape[1])

        col, row = divmod(i, 3)
        axes[row, col].imshow(img_rgb)
        axes[row, col].set_title(f"Quality {q}, Bitrate: {bitrate:.2f}")
        axes[row, col].axis("off")

    plt.tight_layout()
    plt.savefig("images_to_evaluate.png")
    plt.show()

image_col = cv2.imread("original.png")

image_R = image_col[:, :, 2]
image_G = image_col[:, :, 1]
image_B = image_col[:, :, 0]

hist_R, H_R = get_histogram_and_entropy(image_R)
hist_G, H_G = get_histogram_and_entropy(image_G)
hist_B, H_B = get_histogram_and_entropy(image_B)
print(f"H(R) = {H_R:.4f} \nH(G) = {H_G:.4f} \nH(B) = {H_B:.4f} \nH_śr = {(H_R+H_G+H_B)/3:.4f}")

show_rgb_channels(image_R, image_G, image_B)

plot_rgb_histograms(hist_R, hist_G, hist_B)
```

```
image_yuv = cv2.cvtColor(image_col, cv2.COLOR_BGR2YCrCb)

image_Y = image_yuv[:, :, 0]
image_U = image_yuv[:, :, 1]
image_V = image_yuv[:, :, 2]

hist_Y, H_Y = get_histogram_and_entropy(image_Y)
hist_U, H_U = get_histogram_and_entropy(image_U)
hist_V, H_V = get_histogram_and_entropy(image_V)
print(f"H(Y) = {H_Y:.4f} \nH(U) = {H_U:.4f} \nH(V) = {H_V:.4f} \nH_śr = {(H_Y+H_U+H_V)/3:.4f}")

show_yuv_channels(image_yuv)

plot_yuv_histograms(hist_Y, hist_U, hist_V)

qualities = [1, 5, 10, 15, 20, 25, 30, 40, 50, 60, 70, 80, 85, 90, 95, 97, 98, 99, 100]
compare_compression_qualities(image_col, qualities)

qualities_to_evaluate = [1, 5, 15, 25, 50, 95]
show_jpeg_qualities(image_col, qualities_to_evaluate)

cv2.imwrite("compressed_input.png", image_col)
bitrate = 8*os.stat("compressed_input.png").st_size/(image_col.shape[0]*image_col.shape[1])
print(f"Bitrate for PNG: {bitrate:.2f} bpp")
```